Índice espacio-temporal D*R-Tree y sus avances experimentales

María G. Dorzán, Edilma O. Gagliardi
Universidad Nacional de San Luis, Facultad de Ciencias Físico,
Matemáticas y Naturales, Departamento de Informática
San Luis, Argentina, D5700HHW
{ mgdorzan, oli }@unsl.edu.ar

Resumen
Las Bases de Datos Espacio-Temporales permiten almacenar y consultar los cambios de posición, forma y/o tamaño de objetos a lo largo del tiempo. Los métodos espacio-temporales permiten responder consultas que involucran predicados espacio-temporales en forma eficiente, donde los más considerados, en general, son *TimeSlice, Eventos, Intervalo* y *Trayectoria*. En la literatura, podemos encontrar una gran variedad de métodos, los cuales intentan optimizar el desempeño de las consultas, pero siempre apuntando a un subconjunto de las antes mencionadas.

En este artículo presentamos *D*R-Tree*, un índice espacio-temporal, que permite resolver estos cuatro tipos de consultas, sin aumentar la complejidad espacio-temporal.

Presentamos una evaluación experimental más avanzada, que se realizó considerando aspectos propios de una realidad, y que mostró el buen desempeño del índice en aplicaciones de diferentes magnitudes.

Palabras claves: bases de datos espacio-temporales, índices espacio-temporal, consultas espacio-temporal.

1. INTRODUCCIÓN

Actualmente han surgido nuevas aplicaciones que requieren una eficiente manipulación de objetos móviles y de las relaciones existentes entre ellos. Por consiguiente, los sistemas de base de datos deberían ser capaces de soportar estos nuevos tipos de datos y de contar con lenguajes de consultas apropiados. Así, para la recuperación de información, podrían considerarse métodos de acceso espacio-temporales, que surjan como estructuras adicionales o bien índices que resulten de extensiones de los métodos de acceso multidimensional, satisfaciendo de esta manera, las demandas de los usuarios [4, 5, 6, 8].

En este sentido, nuestro trabajo consistió en desarrollar un método de indexación espacio-temporal, D*R-Tree, con el objetivo de evaluar y diseñar estructuras de datos y algoritmos que permitan resolver eficientemente los principales tipos de consultas espacio-temporales tales como *TimeSlice*, *Intervalo*, *Eventos* y *Trayectoria* [1]. Para su diseño y desarrollo, nos basamos en el modelo propuesto en [2], llamado SEST-Index, dado que aprovechamos sus ventajas respecto de una buena utilización de espacio en disco y de su buen tiempo para responder consultas, como así también de la posibilidad que nos permitió ampliar el conjunto de consultas espacio-temporales.

En este artículo presentamos avances experimentales realizados sobre D*R-Tree. La evaluación experimental la realizamos en dos etapas. En la primera, la evaluación consistió en respetar las condiciones de implementación de SEST-Index con el fin de tener los mismos parámetros de comparación. En la segunda etapa, consideramos una evaluación experimental que tuviera más relación con aplicaciones reales, de modo que los parámetros de referencia caracterizaran mejor el comportamiento del índice en situaciones de mayor magnitud. De este modo, la evaluación experimental consolidó la eficiencia del método, a través de los resultados obtenidos, los cuales fueron satisfactorios, mostrando el buen desempeño del índice al responder un conjunto de consultas espacio-temporales, más amplio que el resto de los métodos.

El presente artículo está organizado como sigue: en la sección 2 describimos las estructuras que constituyen el índice D*R-Tree y presentamos los algoritmos de consultas para soportar el procesamiento de las mismas. En la sección 3 mostramos los resultados obtenidos en las dos etapas de la evaluación experimental. Finalmente, damos nuestras conclusiones, y el marco y visión de futuro de nuestro trabajo.

Este trabajo está subvencionado por el Proyecto Tecnologías Avanzadas de Bases de Datos 22/F314, Departamento de Informática, Universidad Nacional de San Luis, y por el Proyecto AL07-PAC-027 Geometría Computacional, de la Universidad Politécnica de Madrid, España.

2. D*R-TREE

D*R-Tree es un método de acceso espacio-temporal que se basa en la idea expuesta en [2]. Sus estructuras permiten almacenar y recuperar objetos espacio-temporales de forma eficiente. Este método está diseñado para responder consultas de tipo TimeSlice, Eventos, Intervalo y Trayectoria. Trata de mantener un equilibrio entre el espacio de disco utilizado por la estructura y el tiempo de acceso empleado en responder los distintos tipos de consulta. En la Figura 1 se muestra el esquema del método de acceso espacio-temporal.

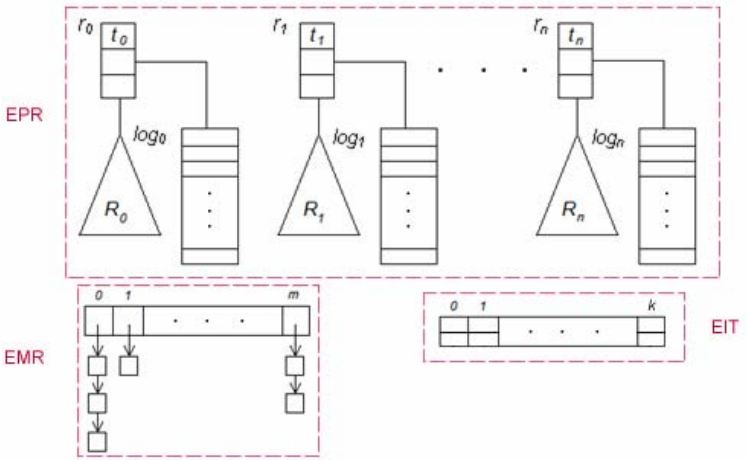

Figura 1: Esquema de D*R-Tree

2.1 ESTRUCTURAS

***Estructura Puntos de Referencia* (*EPR*)**

Almacena instantes de tiempo específicos, llamados puntos de referencia: r_0, r_1,…, r_n. Cada instante de tiempo tiene asociado un *snapshot* de la base de datos. Los puntos de referencia se almacenan en una lista secuencial ordenada por tiempo. Las entradas son tuplas de la forma $r_i = < t_i, R\text{-}root_i, L\text{-}head_i >$, donde t_i corresponde al instante de tiempo asociado al punto de referencia; $R\text{-}root_i$ mantiene la raíz del *i*-ésimo R-Tree [3]; y $L\text{-}head_i$ mantiene la cabecera de la *i*-ésima bitácora. Por tanto, esta estructura tiene asociada una *Colección de R-Trees* (*CRT*) y una *Colección de Bitácoras* (*CB*). Cada snapshot se almacena en un R-Tree, R_i, el cual está asociado a un punto de referencia r_i. Los cambios que ocurren entre puntos de referencia consecutivos, r_i y r_{i+1}, se almacenan en una bitácora, log_i, asociada a un punto de referencia r_i. Cada bitácora es una lista secuencial ordenada por tiempo. Las entradas son tuplas de la forma <*Oid, t*, *Back-log*, *Mbr-act*> donde *Oid* es el identificador del objeto; *t* es el instante de tiempo asociado al movimiento ocurrido; *Back-log* mantiene un puntero a una entrada de una bitácora que almacena el movimiento previo del objeto con identificador *Oid*; y *Mbr-act* mantiene la última posición espacial almacenada del objeto.

***Estructura de Movimientos de Referencia* (*EMR*)**

Esta estructura es una lista secuencial de listas, cuyas entradas del primer nivel poseen la siguiente forma: <*Oid*, *List-head*> donde *Oid* es el identificador del objeto y *List-head* es la cabecera de la lista vinculada l_i. Cada nodo de la lista vinculada l_i tiene la siguiente forma: < *Log-ref*, *Rp-time*, *Next-n*>, donde *Log-ref* es un puntero a la entrada de una bitácora donde se almacena el último movimiento del objeto con identificador *Oid*; *Rp-time* es el instante de tiempo asociado al punto de referencia donde se mantiene la bitácora indicada por *Log-ref*; y *Next-n* es el puntero al siguiente nodo de la lista. Para cada objeto o_i se mantiene un acceso al último movimiento de o_i almacenado en cada bitácora. Se debe tener en cuenta que existe, como máximo, un nodo asociado a cada bitácora y una bitácora está asociada, como máximo, a un solo nodo.

Estructura de Índice de Tiempo (EIT)

Para cada instante de tiempo t_i se mantiene un acceso al primer movimiento registrado en el instante t_i en la bitácora correspondiente. Los instantes de tiempo son almacenados en una lista secuencial ordenada por tiempo. Esta estructura permite encontrar cualquier instante de tiempo en diferentes bitácoras. Las entradas son tuplas de la siguiente forma < *t*; *Log-ref* >, donde *t* es un instante de tiempo y *Log-ref* es un puntero a la entrada de una bitácora donde se almacena el primer movimiento ocurrido en el instante de tiempo *t*.

2.2 CONSULTAS

El tipo de *Consultas de selección* es de la forma "*encontrar todos los objetos que se encontraban en un área o punto específico, durante un intervalo o instante de tiempo específico*". Las consultas típicas son: i) *TimeSlice*: responde consultas expresadas de la siguiente forma: "*encontrar los objetos que se encontraban en un área específica en un instante de tiempo dado*"; ii) *Intervalo*: responde consultas expresadas de la siguiente forma: "*encontrar los objetos que se encontraban en un área específica en un intervalo de tiempo dado*". El tipo de consulta *Eventos* es de la forma: "*recuperar los eventos que ocurrieron en una región en un instante de tiempo dado*". La consulta específicamente requiere recuperar los objetos que entraron o salieron de alguna región de consulta en un instante de tiempo dado. Finalmente, el tipo de consulta *Trayectoria* es de la forma: "*recuperar un conjunto de posiciones espaciales por las cuales ha pasado un objeto en un intervalo de tiempo dado*" [6, 7, 9]. A continuación mostramos los algoritmos diseñados para responder los tipos de consulta descriptos anteriormente:

Algoritmo *TimeSlice* (*R, t*)
Entrada: *R* es el rectángulo de consulta y *t* es el instante sobre el cual se realiza la consulta.
Salida: *Q* es el conjunto de objetos que satisfacen la consulta.
1. $i \leftarrow FindTime(t)$
2. $Q \leftarrow FindInRTree(EPR(i).R\text{-}root, R)$
3. While $EPR(i).t \leq t$ do
4. $\quad Q \leftarrow UpdateQwithLog(EPR(i).L\text{-}head, R, t)$
5. endWhile
6. Return *Q*

Algoritmo *Intervalo* (R, t_o, t_f)
Entrada: *R* es el rectángulo de consulta, t_o es el límite inferior del intervalo de tiempo y t_f es el límite superior sobre el cual se realiza la consulta.
Salida: *Q* es el conjunto de objetos que satisfacen la consulta.
1. $i \leftarrow FindTime(t_o)$
2. $Q \leftarrow FindInRTree(EPR(i).R\text{-}root, R)$
3. While $EPR(i).t \leq t_o$ do
4. $\quad Q \leftarrow UpdateQwithLog(EPR(i).L\text{-}head, R, t_o)$
5. endWhile
6. If todas las entradas de la bitácora $EPR(i).L\text{-}head$ fueron procesadas then
7. $\quad i \leftarrow i + 1$
8. endIf
9. While $EPR(i).t \leq t_f$ do
10. $\quad Q \leftarrow UpdateQwithLog(EPR(i).L\text{-}head, R, t_f)$
11. $\quad i \leftarrow i + 1$
12. endWhile
13. Return *Q*

Algoritmo *Eventos* (*R*, *t*)
Entrada: *R* es el rectángulo de consulta y *t* es el instante sobre el cual se realiza la consulta.

Salida: *Q* es el conjunto de objetos que satisfacen la consulta.
1. $E \leftarrow FindInLogByT(EIT, t)$
2. For each $e \in E$ do
3. If *e.Mbr_act* está en $R \wedge$ *e.Back-log* no está en R then
4. $Q \leftarrow Q \cup \{e\}$ /* entró a R*/
5. else
6. If *e.Mbr_act* no está en $R \wedge$ *e.Back-log* está en *R* then
7. $Q \leftarrow Q \cup \{e\}$ /* salió de *R* */
8. endIf
9. endIf
10. endFor
11. Return *Q*

Algoritmo *Trayectoria* (*Oid*, t_i, t_k)
Entrada: *Oid* es el identificador del objeto a consultar, t_i es el límite inferior del intervalo de tiempo y t_k es el límite superior sobre el cual se realiza la consulta.
Salida: *Q* es el conjunto de objetos que satisfacen la consulta.
1. $c \leftarrow EMR(Oid).List\text{-}head$
2. While $t_k \leq c.Rp\text{-}time$ do
3. $c \leftarrow c.Next\text{-}n$
4. endWhile
5. $b \leftarrow c.Log\text{-}ref$
6. $Q \leftarrow Q \cup \{b.MBR\_act\}$
7. $b \leftarrow b.Back\text{-}log$
8. While $(b.t > t_i) \wedge (b \neq -1)$ do /*si b = -1, se está referenciando al primer R-Tree*/
9. $Q \leftarrow Q \cup \{b.MBR\_act\}$
10. $b \leftarrow b.Back\text{-}log$
11. endWhile
12. If $b = -1$ then
13. $Q \leftarrow FindInicialPosition(Oid)$
14. endIf
15. Return *Q*

Donde:
- *FindTime*(*t*) mediante búsqueda binaria, encuentra el punto de referencia que corresponde a la posición en *EPR* tal que *EPR*(i).t ≤ t.
- *FindInRTree*(*EPR*(*i*).*R-root*, *R*) encuentra los objetos almacenados en el R-Tree con raíz *EPR*(*i*).*R-root* que intersecan *R*.
- *UpdateQwithLog*(*EPR*(*i*).*L-head*, *R*, *t*) actualiza el conjunto de resultado *Q* con los objetos almacenados en la bitácora referenciada por *EPR*(*i*).*L-head*. Los instantes de tiempo de dichos objetos deben ser menor o igual a *t* y su MBR debe intersecar a la región *R*.
- *FindInLogByT*(*EIT*, *t*) utilizando la estructura *EIT* encuentra las entradas de la bitácora que almacena todos los movimientos asociados al instante de tiempo *t*.
- *FindInicialPosition*(*Oid*) encuentra el MBR inicial del objeto con identificador *Oid*.

3. EVALUACIÓN EXPERIMENTAL

*D*R-Tree s*e implementó en lenguaje *Java* utilizando el ambiente de desarrollo Eclipse v3.0.1. Las pruebas se realizaron en una computadora *AMD Athlon 64* de 1.8 Ghz con 1Gb. de RAM, en el sistema operativo *Windows*. Los lotes de prueba se obtuvieron utilizando el generador de datos espacio-temporales GSTD[10]. En las diferentes pruebas se tuvieron en cuenta las siguientes condiciones: la cantidad de objetos en movimiento es fija; el porcentaje de movilidad es alto; los objetos informan los cambios de posición; los objetos se mueven en el plano, en una región acotada y conocida; se utilizan distribuciones de probabilidades para generar los movimientos de los objetos; y el área de las consultas son rectángulos cuyos lados son perpendiculares a los ejes. Cabe aclarar que las pruebas se realizaron para aquellos casos donde la cantidad de entradas necesarias

para almacenar los movimientos producidos en un instante de tiempo no sobrepasa la capacidad de la bitácora. Por tal motivo, algunos experimentos no se muestran ya que no pueden brindar información relevante. La utilización del disco se midió por el número de bloques utilizados por la estructura de datos luego de haber insertado los objetos y sus respectivos movimientos. El tiempo de acceso se definió como el número promedio de bloques leídos luego de haber realizado 100 consultas aleatorias.

3.1 PRIMERA ETAPA

Fue realizada considerando un universo de objetos moviéndose en el plano, con los mismos lotes de prueba utilizados en la evaluación experimental del método base [2]. En esta etapa se respetaron las condiciones de implementación de SEST-Index, evaluando la misma cantidad de objetos, de bloques por bitácora, entre otros. A continuación presentamos los resultados más relevantes; para mayores detalles que los expuestos ver [1]. En las comparaciones se consideró la utilización de almacenamiento y tiempo de acceso a disco en las consultas de tipo TimeSlice, Intervalo, Eventos y Trayectoria.

Esta etapa consistió en evaluar los distintos algoritmos con lotes de prueba de 1000, 3000 y 5000 objetos. Los objetos fueron modelados como puntos en movimiento. Estos puntos fueron distribuidos uniformemente dentro de una región en el instante de tiempo 0.0. Seguidamente, fueron movidos aleatoriamente durante los siguientes 50 instantes hasta alcanzar el instante de tiempo 1.0. Se utilizaron los siguientes porcentajes de movilidad (frecuencia de cambio): 1%, 3%, 5%, 7%, 9%, 11%, 13% y 15% por instante de tiempo. Esto es, que por instante de tiempo se mueve un porcentaje de la cantidad de objetos considerados. Además, se utilizaron bitácoras de 4, 8 y 12 bloques por bitácora con bloques de disco de 1024 bytes.

3.1.1 *Utilización del espacio*

Podemos observar que para ambas estructuras, a medida que aumenta el porcentaje de movilidad mayor es la cantidad de bloques necesarios para almacenarlas. También, observamos que la diferencia en la cantidad de bloques utilizados por ambas estructuras aumenta a medida que se incrementa el porcentaje de movilidad.

La experimentación nos mostró que al aumentar la cantidad de objetos, la cantidad de bloques utilizados aumentó proporcionalmente. También que, mientras la cantidad de bloques por bitácora aumenta, menor es el espacio utilizado por la estructura en general ya que se necesitan mantener menos puntos de referencia. La Figura 2 muestra la cantidad de bloques utilizados en mantener la estructura en ambos métodos con lotes de 5000 objetos, 12 bloques por bitácora y porcentaje de movilidad de 1%, 3%, 5%, 7%, 9% y 11%. La diferencia en el uso del disco, se debe a que nuestra propuesta responde eficientemente a la consulta de tipo Trayectoria, lo que implica el uso de estructuras adicionales y modificaciones en la bitácora. Sin embargo, este costo adicional en el uso de espacio representa un porcentaje bajo adicional respecto de la estructura original, que obviamente resulta poco significativo en comparación a las ventajas obtenidas.

3.1.2 *Consultas*

Para todos los tipos de consulta se realizaron 100 consultas aleatorias. Para el caso de TimeSlice, Intervalo, Evento y Trayectoria se utilizaron rectángulos que cubren un 5% y 10% del área total y la posición del rectángulo y el instante de tiempo de la consulta son aleatorios. Además, para las consultas de tipo Intervalo se utilizaron intervalos de tiempo de tamaño 5 y 10 unidades.

Luego de realizar la experimentación notamos que en D*R-Tree no hay significativas variaciones en el número de bloques leídos, para responder los tipos consulta TimeSlice, Intervalo y Evento con

respecto a SEST-Index ya que ambos índices trabajan de forma similar para resolverlos. Las Figuras 3, 4 y 5 muestran la cantidad de bloques leídos por ambos métodos para responder a los tipos de consulta TimeSlice, Intervalo y Evento con lotes de 5000 objetos, 12 bloques por bitácora, porcentaje de movilidad de 1%, 3%, 5%, 7%, 9% y 11%, y áreas del 5%. Para la consulta de tipo Intervalo se muestran los resultados obtenidos con tamaño de intervalo de 10 unidades.

Cabe aclarar que como *SEST-Index* no responde al tipo de consulta Trayectoria en forma eficiente por no estar diseñada para este fin, se debe encontrar el punto de referencia que contiene el instante correspondiente al límite superior del intervalo de consulta y luego recorrer secuencialmente las bitácoras, en forma descendiente con respecto al tiempo, hasta encontrar el instante correspondiente al límite inferior del intervalo. Como respuesta a la consulta Trayectoria se retornan las posiciones encontradas a lo largo de este recorrido que corresponden al objeto de consulta. *D*R-Tree* simplemente utiliza la estructura adicional *EMR* para encontrar el último movimiento del objeto de consulta en la bitácora correspondiente al límite superior del intervalo. Luego, se recorren las bitácoras utilizando los punteros a una posición de una bitácora en la cual está almacenado el movimiento inmediatamente anterior correspondiente objeto de la consulta. De esta manera, se asegura que sólo se leen los bloques donde existe una referencia a un movimiento del objeto. Además notamos que *SEST-Index* puede no retornar un resultado ya que en intervalos de tiempo donde no se han registrado movimientos del objeto, en el recorrido secuencial no se encontrará información del objeto de consulta. También, se realizaron pruebas donde se compara D*R-Tree con distintos intervalos de tiempo de consulta y se observó que al incrementar el intervalo de consulta, la diferencia entre la cantidad de bloques leídos no es significativa. Los resultados que se muestran en la Figura 6 corresponden a pruebas realizadas para 5000 objetos, con intervalo de 10 unidades de tiempo con 12 bloques por bitácora.

3.2 SEGUNDA ETAPA

La segunda etapa fue motivada por una aplicación real en la cual los parámetros anteriores resultaban limitados. Por lo tanto, la evaluación experimental consistió en utilizar lotes de prueba de 10000 y 20000 objetos. Estos objetos, modelados como puntos moviéndose en el plano, al igual que en la primera etapa, fueron distribuidos uniformemente dentro de una región comenzando en el instante de tiempo 0.0. Estos objetos continuaron moviéndose durante los siguientes 200 instantes hasta alcanzar el instante de tiempo 1.0. Los porcentajes de movilidad utilizados fueron de 1%, 3% y 5%. Notar que al aumentar el porcentaje de movilidad, se incrementa sustancialmente la cantidad de objetos que se mueven en un instante de tiempo. Por tanto, también se requieren bitácoras más grandes. Los tamaños de bitácora usados fueron de 8, 16, 32 y 64 bloques por bitácora con tamaño de bloque de disco de 1024 bytes. En esta segunda etapa las pruebas se realizaron sobre D*R-Tree.

La utilización del disco se midió por el número de bloques utilizados por la estructura de datos luego de haber insertado los objetos y sus respectivos movimientos. El tiempo de acceso se definió como el número promedio de bloques leídos luego de haber realizado 100 consultas aleatorias.

3.2.1 *Utilización del espacio*

Para esta etapa, hicimos la evaluación experimental sólo sobre D*R-tree, observando cuál era el crecimiento de la utilización del espacio en disco. En el caso en que la experimentación consistió en mover 10000 objetos, con porcentajes de movilidad: 1%, 3% y 5%, podemos observar en la Figura 7 en el caso del 1%, cuando se consideran bitácoras de 4 bloques, ocurre que el espacio en disco utilizado es alrededor de los 12500 bloques. Esto se debe a que las bitácoras son pequeñas y se deben determinar más puntos de referencia y en consecuencia existen más árboles R-trees. El caso del 3%, se inicia con bitácoras de 16 bloques, para poder registrar al menos todos los movimientos de un instante de tiempo asociado a un punto de referencia. Como es lo mínimo indispensable para

el tamaño de una bitácora, resulta ser que la cantidad de bloques utilizados también es elevada, superando los 25000 bloques usados. También aquí la justificación es similar.

Para el caso del 5% de movilidad, se requiere iniciar con bitácoras de al menos 32 bloques, con la misma justificación: un instante completo debe poder guardarse en ella; y la cantidad de bloques usados en total alcanza los 20000. Notemos que cuando las bitácoras contuvieron más bloques, entonces los puntos de referencia comenzaron a alejarse temporalmente, lo cual permitió tener árboles RTrees más distanciados en el tiempo y bitácoras que eventualmente almacenaran los movimientos correspondientes a varias instancias. Así, cuando las bitácoras son de 64 bloques, podemos observar que la cantidad de bloques utilizados se reduce sustancialmente, indicando que el número adecuado en estos casos ronda en bitácoras de dicho tamaño.

En el caso en que la experimentación consistió en mantener 20000 objetos, con porcentajes de movilidad: 1% y 3% podemos observar la Figura 8. En el caso del 1%, cuando se consideran bitácoras de 4 bloques, ocurre que el espacio en disco utilizado es alrededor de los 12500 bloques, tal como se vio en el caso anterior. También, se debe a que las bitácoras son pequeñas y se deben determinar más puntos de referencia y en consecuencia hay más árboles R-Trees. El caso del 3%, la cantidad de bloques asciende a 22000, con bitácoras de 64 bloques, ya que es lo mínimo necesario. Claramente, se puede observar, con casos anteriores, con bitácoras más grandes, se reduce significativamente la cantidad de bloques utilizados.

3.2.2 *Consultas*

Para la consulta TimeSlice, la Figura 9 nos muestra para 10000 objetos con porcentajes de movilidad 1%, 3%, 5% y 7%, la cantidad de bloques leídos, donde el área de la consulta es del 5% del área total. Al igual que en la primera etapa de la evaluación experimental, esta consulta tiene un comportamiento heterogéneo en el sentido de cómo se determinan las curvas. Podemos observar que para el 1% de movilidad, con bitácoras de 16 bloques se necesitan leer menos de 15 bloques, para el 3% alrededor de 16 o 17 bloques y crece más en el caso del 5%, donde se leen cerca de 24 bloques. Nuevamente, tiene que ver con el tamaño de las bitácoras, puesto que requiere buscar en bitácoras el instante de tiempo dado y los puntos de referencia están más distanciados.

En el caso del 3%, se da una coincidencia entre bitácoras de 16 y 32 bloques porque en ambos casos se almacenan la misma cantidad de instantes de tiempo, habiendo una con más espacio desperdiciado (porque llega el nuevo punto de referencia y hay que construir el R-Tree asociado, dejando atrás la bitácora). En el caso del 3% con bitácoras de 64 bloques, se observa el crecimiento de la cantidad de bloques leídos, ya que almacena varios instantes de tiempo. Cuando el porcentaje de movilidad es 5%, con bitácoras de 64 bloques, baja la cantidad de bloques leídos respecto del caso anterior, y esto se debe a que la bitácora almacena menos instantes de tiempo. En los casos del 1% y 3%, se mantiene lo mismo. En el caso de movilidad del 7%, con bitácoras de 64 bloques, ocurre similar al caso del 3% de movilidad con bitácoras de 64 bloques.

Dado los resultados de la consulta TimeSlice (Figura 10), para 20000 objetos con porcentajes de movilidad 1% y 3%, donde el área de la consulta es del 5% del área total, la cantidad de bloques leídos disminuye. Lo interesante de este caso, es que con una población de 10000 objetos, con porcentajes de movilidad similares, se mantuvo estable respecto de la cantidad de bloques leídos.

Para la consulta Intervalo, podemos observar que mientras más grande es el porcentaje de movilidad, mayor es el número promedio de bloques leídos, ya que para cada instante de tiempo se necesitan almacenar mayor cantidad de entradas en la bitácora. Por la misma razón, para mayor cantidad de objetos, mayor es la cantidad de bloques leídos. Si bien el tamaño de la bitácora influye en el espacio utilizado para almacenar la estructura como mencionamos anteriormente, no afecta en

la cantidad de bloques leídos para responder este tipo de consulta. Notamos que no hay variaciones significativas entre los resultados obtenidos para 10000 y 20000 objetos.

Respecto de Eventos, observamos que mientras más grande es el porcentaje de movilidad, mayor es el número promedio de bloques leídos, ya que en cada instante de tiempo ocurren más eventos. La cantidad de bloques por bitácora no disminuye el rendimiento de la estructura. Cabe aclarar que el área de consulta no influye en los resultados ya que en todos los casos, debemos examinar todos los movimientos que se produjeron en el instante consultado.

Para Trayectoria, notamos que el porcentaje de movilidad y la cantidad de objetos no afecta el desempeño del algoritmo que responde a este tipo de consulta ya que el diseño de la estructura permite acceder sólo a aquellas posiciones donde el objeto ha registrado un movimiento.

Como muestran las siguientes figuras, se mantiene casi estable la cantidad de bloques leídos, aunque el porcentaje de movilidad y la cantidad de objetos varíen.

CONCLUSIONES

En este artículo presentamos D*R-Tree, un índice espacio-temporal, que integra la resolución de cuatro tipos de consultas espacio-temporales, sin aumentar la complejidad espacio-temporal. Presentamos su estructura, los algoritmos de consulta y la evaluación experimental realizada, la que mostró el buen desempeño del índice en aplicaciones de diferentes magnitudes respecto de la población de objetos en movimiento.

Durante las dos etapas de la evaluación experimental, notamos que una variable importante en la evaluación experimental de D*R-Tree es el tamaño de la bitácora, ya que éste determina no sólo que tan grande puede llegar a ser el espacio utilizado por la estructura, sino también el tiempo de acceso que se utilizará para responder las consultas. Para determinar el tamaño de bitácora adecuado que se ajuste mejor a nuestras necesidades, debemos tener en cuenta que a medida que el tamaño de la bitácora aumenta, menor será la utilización del espacio. Esto se debe a que la estructura en general mantendrá menos R-Trees.

Sin embargo, el hecho de tener bitácoras más grandes fuerza a aumentar la cantidad de búsquedas secuenciales dentro de las bitácoras, deteriorando consultas del tipo TimeSlice e Intervalo ya que en ambos casos, la cantidad promedio de bloques leídos sería mayor.

El valor indicado para las dimensiones de la bitácora tiene que tener en cuenta ambos aspectos.

Concluimos que nuestra estructura resulta una contribución al área de investigación, ya que proponemos un método de acceso espacio-temporal que responde eficientemente los principales tipos de consulta: TimeSlice, Intervalo, Eventos y Trayectoria.

ANEXO FIGURAS

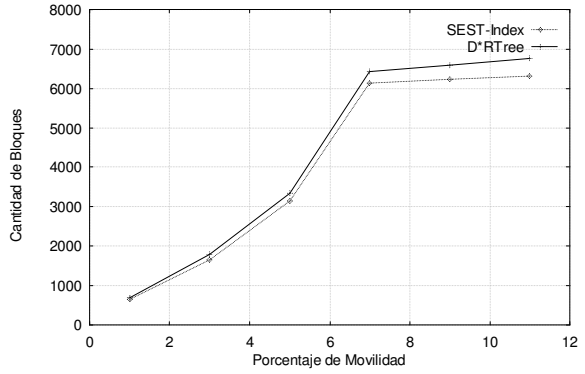

Figura 2: Número de bloques utilizados para 5000 objetos (12

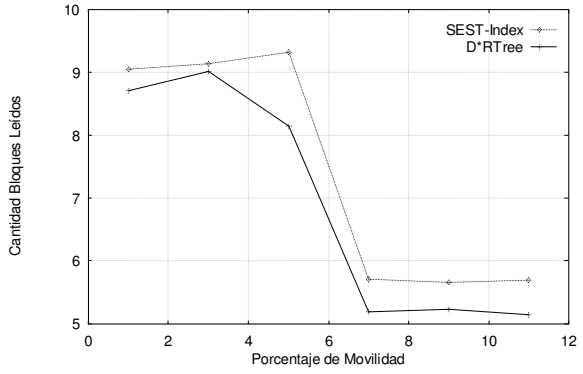

Figura 3: Número de bloques leídos para TimeSlice en un

bloques por bitácora)

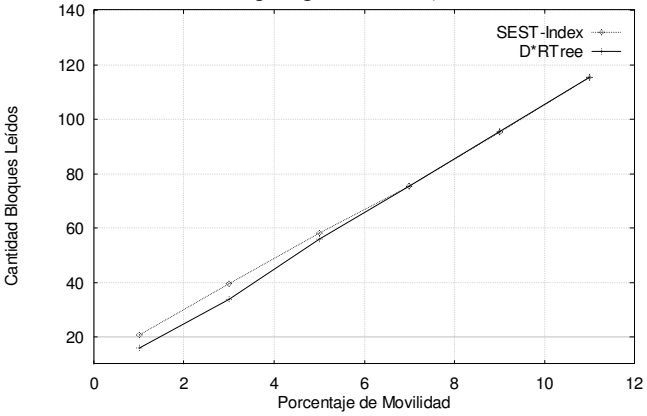

Figura 4: Número de bloques leídos para Intervalo de tamaño 10 en un área de 5% para 5000 objetos (12 bloques por bitácora)

área de 5% para 5000 objetos (12 bloques por bitácora)

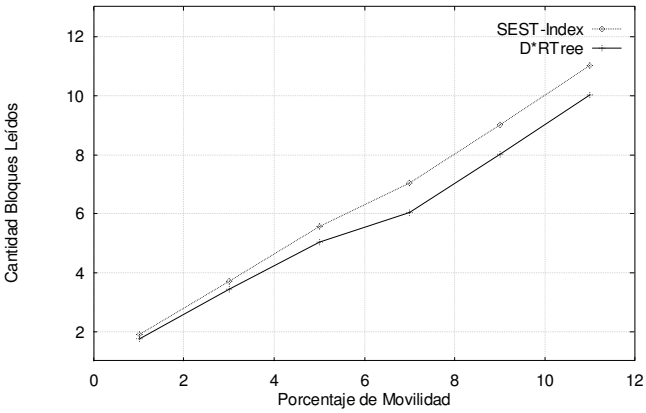

Figura 5: Número de bloques leídos para Eventos en un área de 5% para 5000 objetos (12 bloques por bitácora)

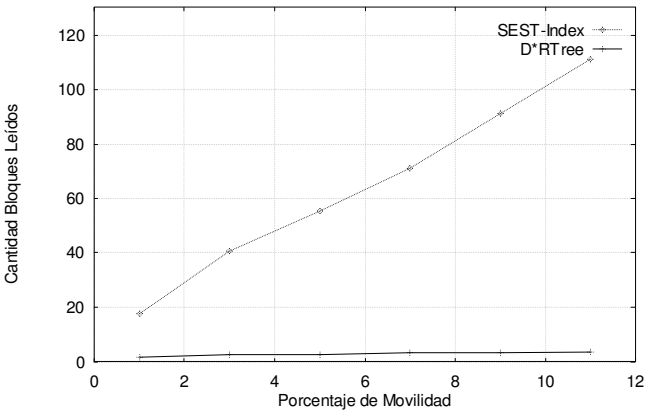

Figura 6: Número de bloques leídos para Trayectoria con intervalo de 10 unidades de tiempo para 5000 objetos (12 bloques por bitácora)

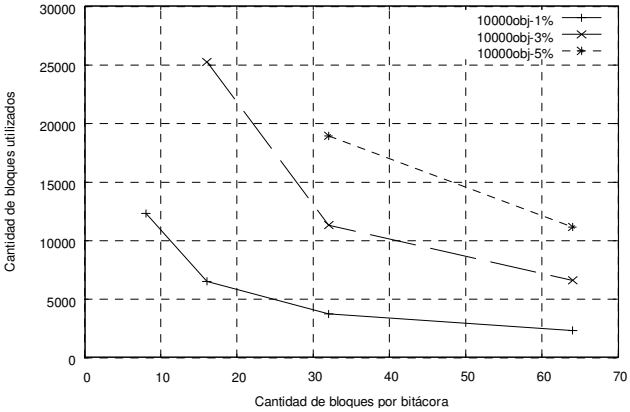

Figura 7: Número de bloques utilizados para 10000 objetos

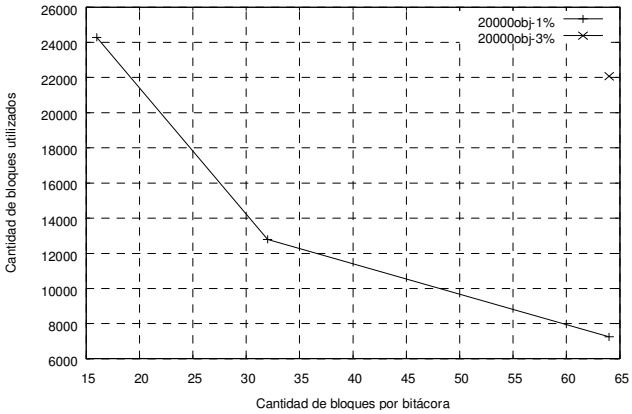

Figura 8: Número de bloques utilizados para 20000 objetos

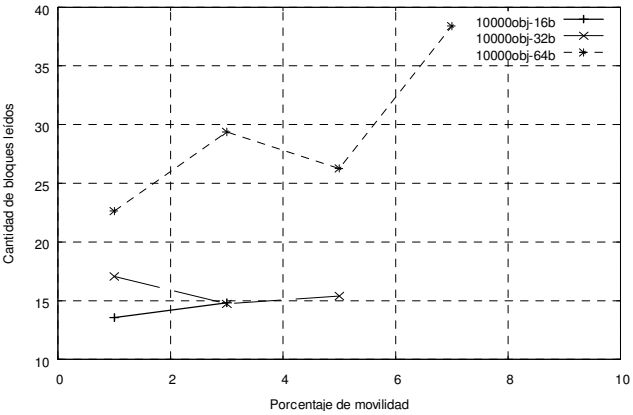

Figura 9: Número de bloques leídos para TimeSlice en un área de 5% para 10000 objetos

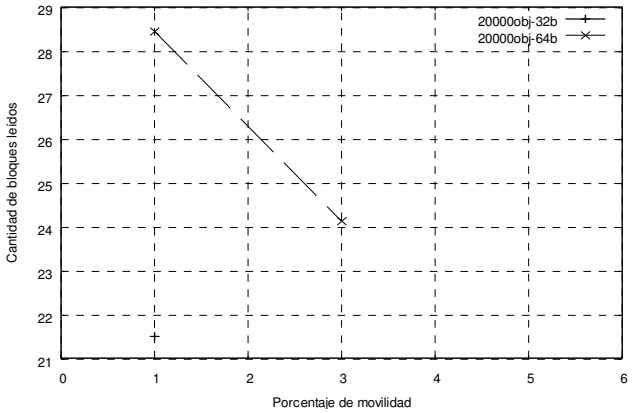

Figura 10: Número de bloques leídos para TimeSlice en un área de 5% para 20000 objetos

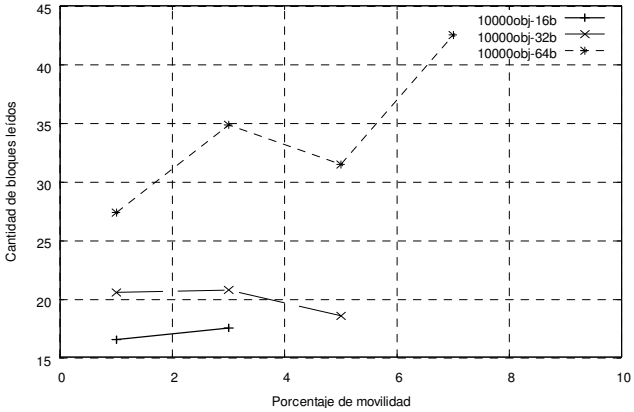

Figura 11: Número de bloques leídos para TimeSlice en un área de 10% para 10000 objetos

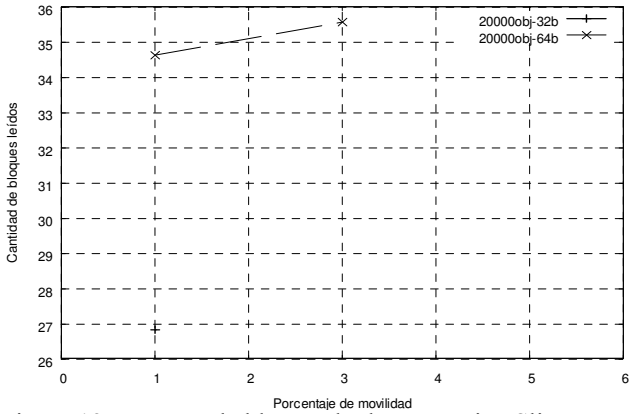

Figura 12: Número de bloques leídos para TimeSlice en un área de 10% para 20000 objetos

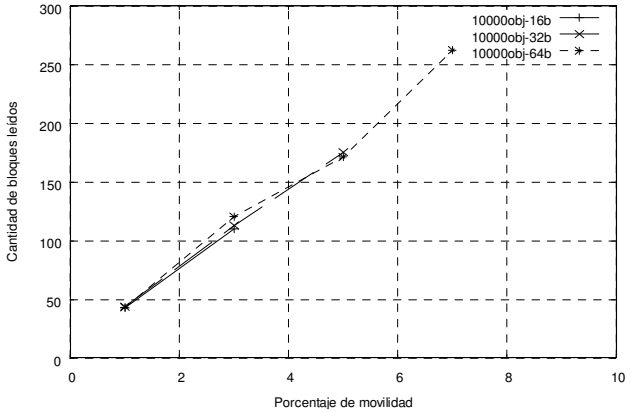

Figura 13: Número de bloques leídos para Intervalo en un área de 5% para intervalos de 5 unidades para 10000 objetos

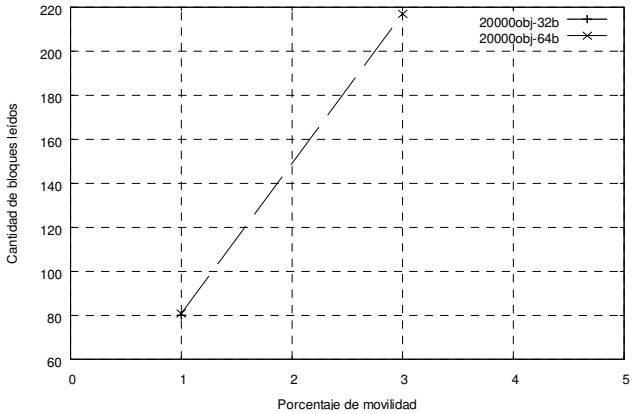

Figura 14: Número de bloques leídos para Intervalo en un área de 5% para intervalos de 5 unidades para 20000 objetos

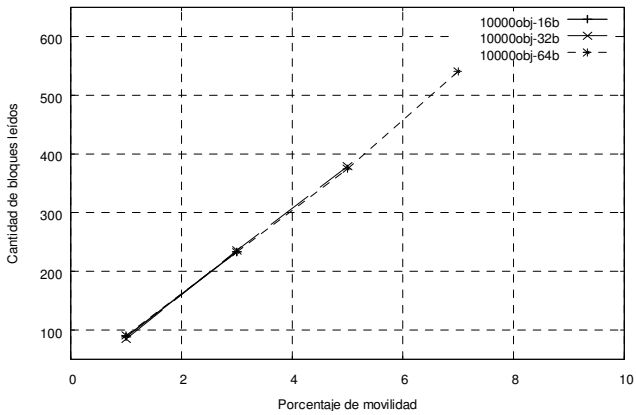

Figura 15: Número de bloques leídos para Intervalo en un área de 10% para intervalos de 10 unidades para 10000 objetos

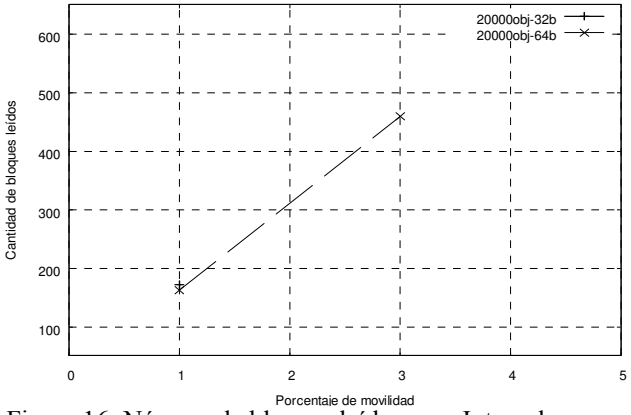

Figura 16: Número de bloques leídos para Intervalo en un área de 10% para intervalos de 10 unidades para 20000 objetos

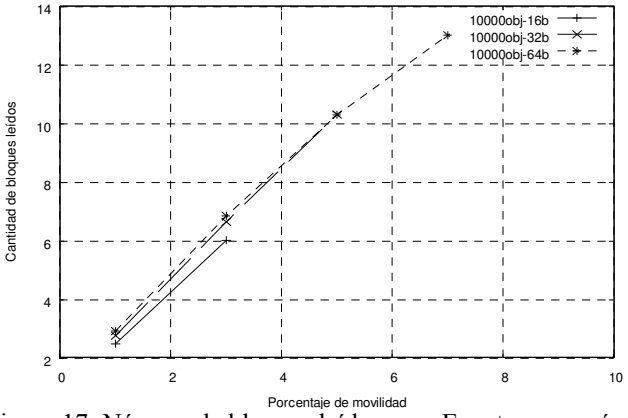

Figura 17: Número de bloques leídos para Eventos en un área de 5% para 10000 objetos

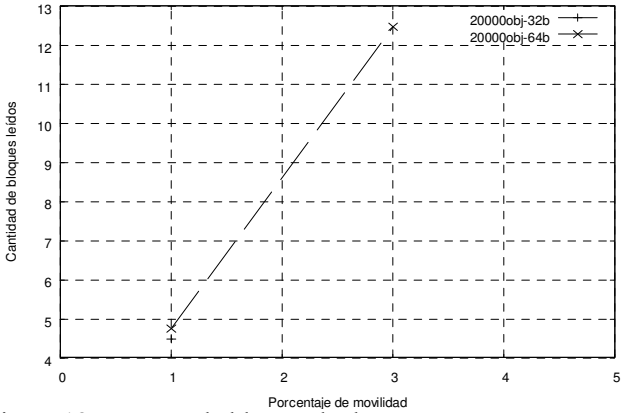

Figura 18: Número de bloques leídos para Eventos en un área de 5% para 20000 objetos

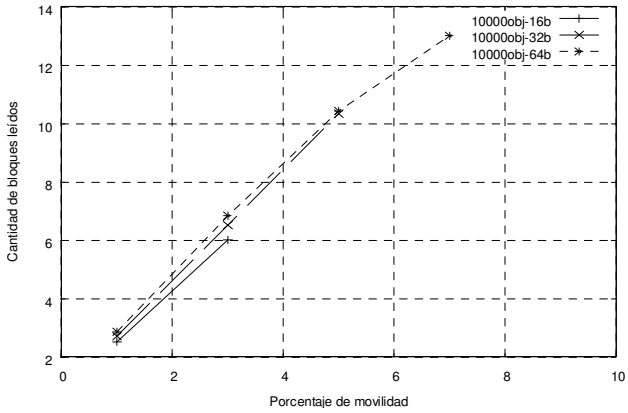

Figura 19: Número de bloques leídos para Eventos en un área de 10% para 10000 objetos

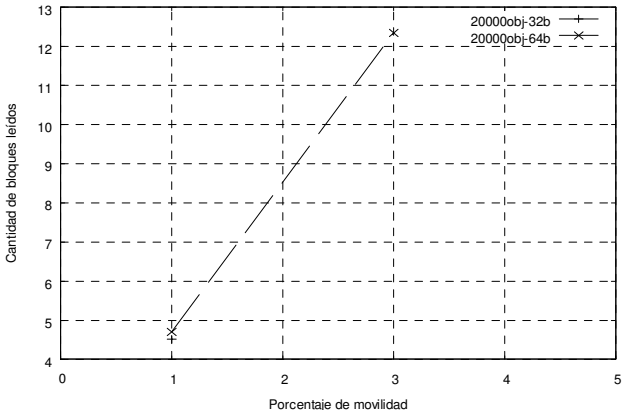

Figura 20: Número de bloques leídos para Eventos en un área de 10% para 20000 objetos

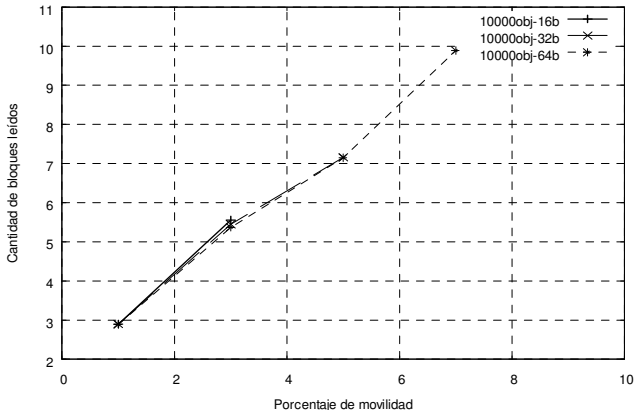

Figura 21: Número de bloques leídos para Trayectoria en un área de 5% para intervalos de 5 unidades para 10000 objetos

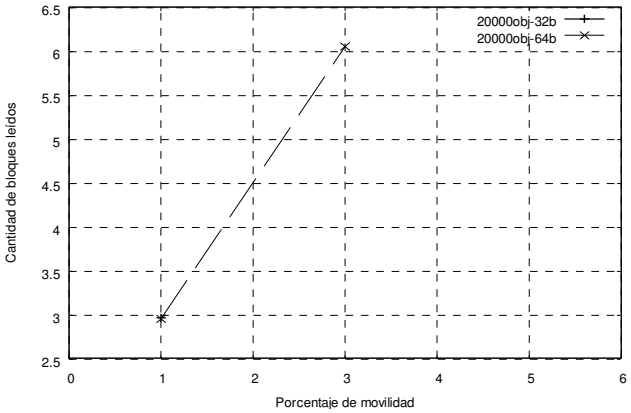

Figura 22: Número de bloques leídos para Trayectoria en un área de 5% para intervalos de 5 unidades para 20000 objetos

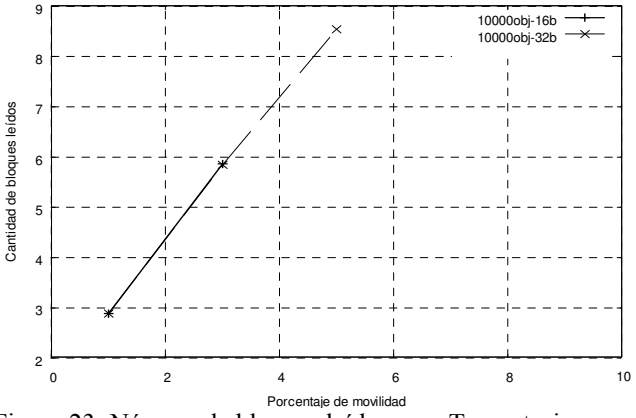

Figura 23: Número de bloques leídos para Trayectoria en un área de 10% para intervalos de 10 unidades para 10000 objetos

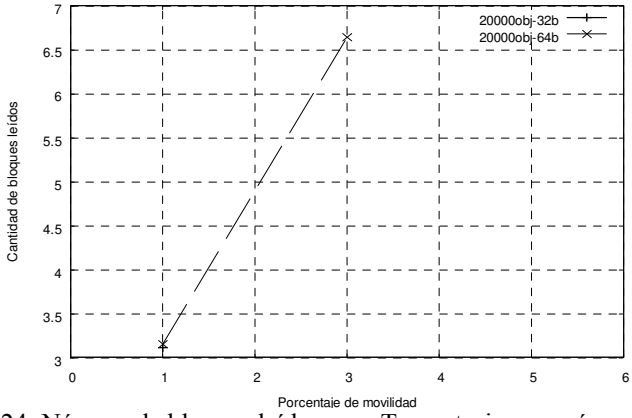

Figura 24: Número de bloques leídos para Trayectoria en un área de 10% para intervalos de 10 unidades para 20000 objetos

REFERENCIAS BIBLIOGRÁFICAS

[1] Dorzán M., Gagliardi E., Gómez Barroso J., y Gutiérrez Retamal G. D*R-Tree: un método eficiente para responder consultas espacio-temporales. CACIC. (2006)

[2] Gutiérrez G. A Spatiotemporal Access Method based on Snapshots and Events. ACM GIS'05, Bremen, Germany. (2005)

[3] Guttman, A. R-Trees: A dynamic index structure for spatial searching. In ACM SIGMOD Conf. on Management of Data, pages 47-57, Boston, ACM. (1984)

[4] Güting, R.H., An introduction to Spatial Database System. VLDB Journal (1994)

[5] Mokbel M., Ghanem T., Aref W. Spatio-temporal Access Methods, IEEE Data Engineering Bulletin 26, pp. 40-49. (2003)

[6] Pfoser D., Jensen C., and Theodoridis Y. Novel Approaches in Query Processing for Moving Object Trajectories. In Proc. of the Intl. Conf. on Very Large Data Bases, VLDB, pages 395–406, (2000)

[7] Porkaew K., Lazaridis I., and Mehrotra S.. Querying Mobile Objects in Spatio-Temporal Databases. In Proc. of the Intl. Symp. on Advances in Spatial and Temporal Databases, SSTD, (2001)

[8] Samet H. Foundations of Multidimensional and Metric Data Structures. ISBN-10: 0123694469. Morgan Kaufmann (2006)

[9] Theodoridis Y., Vazirgiannis M., and Sellis T. Spatio-Temporal Indexing for Large Multimedia Applications. In Proc. of the IEEE Conference on Multimedia Computing and Systems, ICMCS, (1996)

[10] Theodoridis Y., Silva J., and Nascimento M..On the generation of spatiotemporal datasets. (SSD '99), 1999.